BLACK CODHER

CODING PROGRAMME

CODING BLACK FEMALES

Black Codher Bootcamp

NÍ YÓ

BLACK CODHER

CODING PROGRAMME

CODING BLACK FEMALES

UNIT 3
The DOM

NÍ YÓ

RECAP

- while loops
- for loops
- Arrays
- Arrays with loops

WHAT YOU'LL BE LEARNING TODAY?

BLACK CODHER
CODING PROGRAMME

How to modify the **DOM** with our new skills

WHAT YOU'LL NEED

The VSCode **Live Server** extension

Once you have it installed, you're in the *session4/* folder AND you've pressed the *Go Live* button in the bottom right corner, this is where you'll need to go to in the browser:
http://127.0.0.1:5500/unit03-javascript/session4/

THINGS ARE A LITTLE DIFFERENT

BLACK CODHER
CODING PROGRAMME

We've provided you with some code to help you in this session. If we put it all in the `index.js` that file would become confusing and messy.

So we've split the code into separate files and connected them to the `index.js`. Don't worry about them for now...we'll be learning about **modules** later

BLACK CODHER

CODING PROGRAMME

THE DOM

WHAT IS THE DOM?

DOM stands for **D**ocument **O**bject **M**odel.

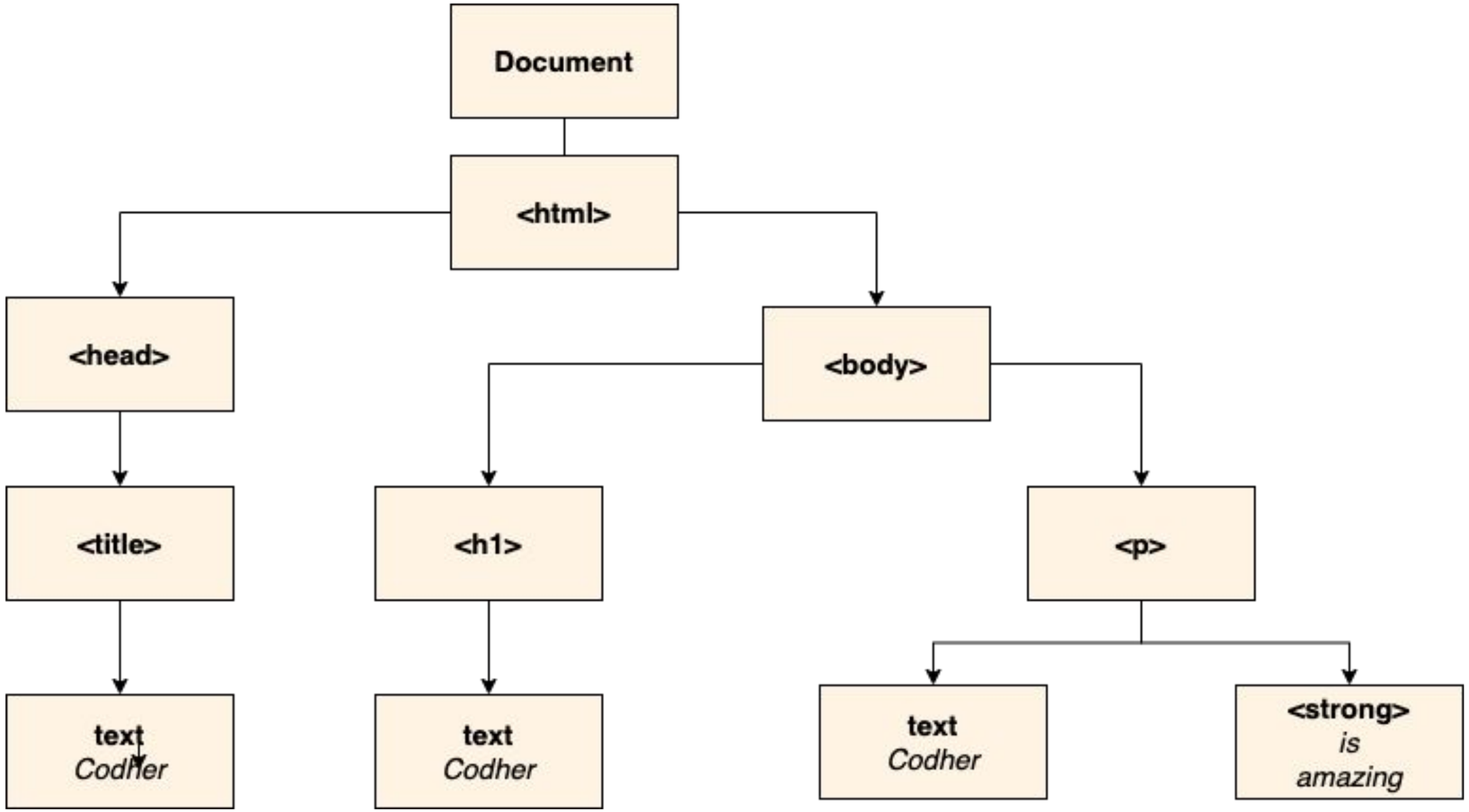
When the **HTML** is loaded on our page, the browser generates its **DOM**. This enables us to access and interact with **HTML** elements using **JavaScript**.

DOM REPRESENTATION

This is a reminder of how the **DOM** is represented in HTML. You should be familiar with this from the HTML section of the bootcamp

```html
<html>
  <head>
   <title>Codher!</title>
  </head>
  <body>
   <h1>Codher</h1>
   <p>
     Codher <strong> is amazing</strong>
   </p>
  </body>
</html>
```

DOM REPRESENTATION



INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

The **DOM** is represented by **nodes**. Each node has different properties, and nodes are connected (like in the tree previous diagram).

Now it's time to interact with the **DOM** of our page and get the parent node of the body using the browser's console.

```
console.log(document.body.parentNode);
```

INTERACTING WITH THE DOM

For this example we used the browser's console because as the script gets loaded before the body, the element will not be there if we simply added this to our `index.js`

In order to use our `index.js` we will be wrapping the `console.log` into a function and modifying it to display all the DOM children elements:

```
function listDomElements() {
  const children = document.body.childNodes

  for (let i = 0; i < children.length; i = i + 1) {
    console.log(children[i])
  }
}
```

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

To trigger this function let's add call it with an `onclick` on an anchor `<a>` tag:

```
<a href="#" onclick="listDomElements()">List DOM elements</a>
```

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

Fortunately, interacting with the DOM doesn't need to be this complicated. An easier way is to retrieve elements using their tags:

```
document.getElementsByTagName('h1');
```

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

This approach is not ideal as you may have multiple paragraphs, divs, links or other elements.

The most common approach is to retrieve an element by its **id**.

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

Let's add a `description` **id** to the paragraph (`<p>`) element and then retrieve it in JavaScript:

```
const description = document.getElementById('description');
console.log(description.innerHTML);
```

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

Hopefully by now you have noticed that in JavaScript there are many ways to retrieve information and store it in a **variable**.

According to MDN docs, the "*recommended modern approach*" to selecting a single element is using `document.querySelector()`. It allows you to get an element using any **CSS selector**. If you use a **class** name, it will return the first element with that name.

```
const description = document.querySelector('#description');
console.log(description.innerHTML);
```

INTERACTING WITH THE DOM

BLACK CODHER
CODING PROGRAMME

If you want to interact with multiple elements at once the recommended method is `document.querySelectorAll()`.

This method returns a `NodeList` representing a list of the document's elements that match the specified group of selectors.

Just like with the `document.querySelector()` you can pass any **CSS Selector** to the `document.querySelectorAll()` method

TASK

In the `index.html` there 3 commented `<p>` elements with the **class** name `about`.

Uncomment all 3 `<p>` elements

- Return the `innerHTML` of the first element
- Return the `NodeList` that represents all the `<p>` elements with the **class** name `about`.

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

GREEN - I feel comfortable with everything you've said

BLACK CODHER

CODING PROGRAMME

MODIFYING HTML

CREATING NEW HTML ELEMENTS

BLACK CODHER
CODING PROGRAMME

There are steps we need to follow to create new elements:

```
// 1. creating an element
document.createElement('<tagName>');

// 2. creating text nodes
document.createTextNode('<text>');

// 3. adding children to elements
document.appendChild('<node>');
```

CREATING NEW HTML ELEMENTS

BLACK CODHER
CODING PROGRAMME

Using the `birmingham` object in the `birmingham.js` file, we'll display the population of the city by creating a `displayPopulation` function:

```
function displayPopulation() {
  // Make a new <p></p> for population. This is not attached to the DOM yet.
  const paragraph = document.createElement('p');

  // Make some text content to put into your <p></p>
  const content = document.createTextNode('Population: ' + birmingham.population);

  // Put the text content into the <p></p>.
  paragraph.appendChild(content);

  // Finally the population can be appended to the body, and will become visible in the browser.
  document.body.appendChild(paragraph);
}
```

TASK

In the `index.js` create a new `button` in JavaScript that calls the `displayPopulation`

- Create a new button in JavaScript (call it `button`)
- Create a text node. The text should say *Show Population*
- Append the text to the button
- Add an `onclick` to the button (`button.onclick = displayPopulation`)
- Append the `document.body` with the new button

The `displayPopulation` function from the `birmingham.js` is already connected to the `index.js`. You can call it as if you wrote the function inside the `index.js`. Use `displayPopulation` for guidance AND the code you write doesn't have to be in a function

CREATING MULTIPLE ELEMENTS WITH LOOPS

BLACK CODHER
CODING PROGRAMME

Previously we learnt about arrays and loops.

Maybe we could use our new skills to create multiple HTML elements at once? 🤔

Inside the `giveMeFruits.js` there's a `createListOfFruits` function that takes an **array** of fruit as its parameter and **renders** them in an unordered list (`<ul>`) in the browser. The unordered list (`<ul>`) will be placed inside a `<div>` with the **id** `list-container`.

Let's breakdown this function...

CREATING MULTIPLE ELEMENTS WITH LOOPS

```
function createListOfFruits(fruits) {
  const listContainer = document.querySelector('#list-container');

  const title = document.createElement('h2');
  const titleText = document.createTextNode('List of fruit');
  const list = document.createElement('ul');

  fruits.map((fruit) => {
    const listItem = document.createElement('li');
    listItem.textContent = fruit;
    list.appendChild(listItem);
  });

  title.appendChild(titleText);
  listContainer.appendChild(title);
  listContainer.appendChild(list);
}
```

1. Our **function** parameter is an **array** of fruits
2. Inside the **function** body the first thing to do is get the element we want add our list to (`listContainer`)
3. Create a `<h2>` for our title
4. Create a text node (the actual text of the title)
5. Create a `<ul>`

CREATING MULTIPLE ELEMENTS WITH LOOPS

```
function createListOfFruits(fruits) {
  const listContainer = document.querySelector('#list-container');

  const title = document.createElement('h2');
  const titleText = document.createTextNode('List of fruit');
  const list = document.createElement('ul');

  fruits.map((fruit) => {
    const listItem = document.createElement('li');
    listItem.textContent = fruit;
    list.appendChild(listItem);
  });

  title.appendChild(titleText);
  listContainer.appendChild(title);
  listContainer.appendChild(list);
}
```

Now the fun begins! We want to create a `<li>` for each fruit in our array

1. Loop through each all the fruits using the `map` **method**
2. For each fruit in the array we create a `<li>`. This is done using the `document.createElement('li')` **method**
3. The actual name of the fruit is added to the `<li>` text
4. Each newly created `<li>` (including the name of the fruit) is added to the unordered list (`<ul>`)

CREATING MULTIPLE ELEMENTS WITH LOOPS

```
function createListOfFruits(fruits) {
  const listContainer = document.querySelector('#list-container');

  const title = document.createElement('h2');
  const titleText = document.createTextNode('List of fruit');
  const list = document.createElement('ul');

  fruits.map((fruit) => {
    const listItem = document.createElement('li');
    listItem.textContent = fruit;
    list.appendChild(listItem);
  });

  title.appendChild(titleText);
  listContainer.appendChild(title);
  listContainer.appendChild(list);
}
```

And finally...

1. We're adding the text to our `<h2>` title
2. Adding our title to the DOM
3. Adding our list to the DOM

CREATING MULTIPLE ELEMENTS WITH LOOPS

BLACK CODHER
CODING PROGRAMME

You might be wondering what's the benefit of using a **function** that loops through an array to create the elements, I can just manually add the `<li></li>` for each of them myself.

What would happen if you had a big list of 50 fruit to display? Would you write out the `<li></li>` for each of them? What if you wanted to remove some?

Inside the `index.js` call the **function** (an **array** of fruits called `fruits` has already been imported into the `index.js` for you to pass as the parameter) and see what happens?

TASK

We've changed our mind. We only want to display fruits with a short name.

Add a `filter` inside of the `createListOfFruits` function so you filter out any fruits that have more than 5 letters in their name (hint: you can use the `length` property on a **string** too!)

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

GREEN - I feel comfortable with everything you've said

BLACK CODHER

CODING PROGRAMME

EVENTS

WHAT IS AN EVENT?

BLACK CODHER
CODING PROGRAMME

An **event** is a signal that something has happened.

When a page loads, that's an **event**.

When a user clicks, that's an **event**.

LIST OF EVENTS

Here's a list of the most useful DOM events

Mouse events:

- `click` – when the mouse clicks on an element (touchscreen devices generate it on a tap).
- `contextmenu` – when the mouse right-clicks on an element.
- `mouseover` / `mouseout` – when the mouse cursor comes over / leaves an element.
- `mousedown` / `mouseup` – when the mouse button is pressed / released over an element.
- `mousemove` – when the mouse is moved.

LIST OF EVENTS

BLACK CODHER
CODING PROGRAMME

Keyboard events:

- `keydown` and `keyup` – when a keyboard key is pressed and released.

Form element events:

- `submit` – when the visitor submits a `<form>`.
- `focus` – when the visitor focuses on an element, e.g. on an `<input>`.

Document events:

- `DOMContentLoaded` – when the HTML is loaded and processed, DOM is fully built.

CSS events:

- `transitionend` – when a CSS-animation finishes.

EVENT LISTENERS

We've already used onclick during this session. onclick is a HTML attribute and an event that occurs when a user clicks on an element. However, there is another way to listen out for events: addEventListener

EVENT LISTENERS

We've already used `onclick` during this session. `onclick` is a HTML attribute and an **event** that occurs when a user clicks on an element.

However, there is another way to listen out for events: `addEventListener`

EVENT LISTENERS

BLACK CODHER
CODING PROGRAMME

The `addEventListener()` is an inbuilt function which takes the event to listen for, and a second argument to be called whenever the described event gets fired. Any number of event handlers can be added to a single element without overwriting existing event handlers.

Parameters:

- **event:** event can be any valid JavaScript event.Events are used without `on` prefix like use `click` instead of `onclick` or `mousedown` instead of `onmousedown`.
- **listener(handler function) :** It can be a function which respond to the event occur.

EVENT LISTENERS

In the `addEvents.js` there's an example of three events:

- `click`
- `mouseover`
- `mouseout`

We'll explain what is happening in the next slide, but take a few minutes to play with it and see what happens.

All you need to do is copy everything from `addEvents.js`, paste into `index.js` and in the `index.html` uncomment the

- `<button>` with the **id** `clickMe`
- `<p>` with the **id** `hovering`
- `<p>` with the **id** `effect`

EVENT LISTENERS

```
const x = document.querySelector('#clickMe')
const y = document.querySelector('#hovering')

x.addEventListener('click', respondClick)
y.addEventListener('mouseover', respondMouseOver)
y.addEventListener('mouseout', respondMouseOut)

function respondMouseOver() {
  document.querySelector('#effect').innerHTML += 'MouseOver Event' + '<br>'
}

function respondMouseOut() {
  document.querySelector('#effect').innerHTML += 'MouseOut Event' + '<br>'
}

function respondClick() {
  document.querySelector('#effect').innerHTML += 'Click Event' + '<br>'
}
```

In this example two events `mouseover` and `mouseout` are added to the same element.

If the text is hovered over then `mouseover` event occur and `respondMouseOver` function invoked, similarly for `mouseout` event `respondMouseOut` function invoked.

EVENT LISTENERS

If we can add event listeners, surely we can remove them also? Fortunately we have the `removeEventListener` method.

This method will remove event handlers that was previously attached using the `addEventListener()` method.

Imagine we have a mouseover effect that is annoying and we just want it to stop...

TASK

We have a big `<div>` and everytime we hover over it a random number appears in a `<p>` underneath. We want this behaviour to stop!

- in the `index.html` uncomment the following:
  - `<div>` with the **id** `remove-handler`
  - the `<p>` and `<button>` inside the `<div>`
  - `<p>` with the **id** `random-number`
- copy the contents of the `removeEvents.js` and paste into `index.js`
- add a click `eventListener` to the `remove-handler-button`. The 2nd parameter should be a `removeHandler` function
- Create a `removeHandler` function that removes the `eventListener` from the `remove-handler` `<div>`

Checkpoint!

How are you feeling?

RED - I have no idea what you're talking about

YELLOW - I have some questions but feel like I understand some things

GREEN - I feel comfortable with everything you've said

BLACK CODHER

CODING PROGRAMME

SUMMARY

SUMMARY

BLACK CODHER
CODING PROGRAMME

- We learnt what the **DOM** is
- How to add new elements to the **DOM**
- Used an array of strings to dynamically create list items
- We learnt about events
- We learnt how to add events
- We learnt how to remove events

BLACK CODHER

CODING PROGRAMME

HOMEWORK

HOMEWORK

BLACK CODHER
CODING PROGRAMME

During the HTML/CSS unit you started making a book shop website. Let's add some JavaScript to it:

- Create an **array** of objects called `books`
  - Each **object** should include the book's `name`, `author`, `price`, `description` and `image`
- Create a **function** that loops through the **array** of books, creates the HTML elements that will display information about each book and renders (displays) the elements in the browser

Don't forget to push your homework to GitHub for practice!